An Efficient Approach for Building Compressed Full-text Index for Structured Data

Jun Liang
Training Center of Electronic Information
Beijing Union University
Beijing, Chinese
liangjun2236@sina.com

Lin Xiao
College of Applied Science and Technolgy
Beijing Union University
Beijing, Chinese
xiaolin.ustc@gmail.com

Di Zhang
Institute of Software
Chinese Acdemic of Science
Beijing,Chinese
zhangdi@mail.rdcps.ac.cn

Abstract*—The self-index is a kind of highly compressed, self-contained full-text index. It is designed for indexing plain texts in order to reduce its permanent storage, as well as to enhance searching performance. Apart from being a sequence of characters, usually the text has specific internal structure. The data record, as a basic model of structured data, is therefore employed to represent and organize such form of data widespread. In this paper, we design and implement an approach to building the self-index for data records via text medium. Our approach indexes the data records through an intermediate text which accommodates aligned record fields by stuffing delimiters among them. By theoretical analysis, we give the upper bounds of permanent space of our approach in a worst case. In addition, we report a series of experimental results to validate the correctness and efficiency of the proposed approach.*

Keywords-self-index; compressed full-text index; data compression; structured data

I. INTRODUCTION

The self-index [7] is a kind of highly compressed full-text index that contains enough information to efficiently reproduce any portion of the text. Without any lexical analysis, such as spacing or stemming, the algorithms of the self-index merely generate a reversible transformation for the text to achieve nearly optimal results on both permanent storage and searching performance. At present, several alternatives are invented to build the self-index for the plain text, mainly including the FM-index, the CSA, and the LZ-index [7]. However, beyond being sequences of characters, a great part of texts are also used as the medium to express structured and semi-structured data as well [4]. The research on building the self-index for such kind of data has not been developed sufficiently yet [7].

The data record is a basic form of organizing data in the relational model [8]. It has widely been used to store structured data due to the domination of the relational model in database field. Because of the substantial connection between the IR and database, it is common for IR systems to represent structured data in form of data records [4]. Thus, studying how to build the self-index for the data records has foundational significance for the problem of self-indexing structured data.

In this paper we design and implement a simple and effective model that builds the self-index for data records. We analyze its expected storage occupancy and performance in theory, and conduct a series of experiments to demonstrate its advantages. Our approach provides a new tradeoff between the permanent space and searching performance.

The rest of paper is organized as follows. First, we briefly review some basic concepts on the entropy of text and the self-index. Then, we present the design and implementation of our approach, together with necessary theoretical analysis. Next, a series of experiments are performed to test the performance of our solution. Finally, we conclude this paper.

II. BACKGROUND

A. Basic Concepts on Entropy of Text

In this section several useful concepts will be introduced to measure the entropy of the text string.

Definition 1. *The* zeroth order empirical entropy [6] *of T over an alphabet set* Σ *is defined as*

$$\hat{H}_0(T) = -\sum_{c\in\Sigma} p(c)\log p(c)$$

Definition 2. *The* k-th order empirical entropy [6] *of T is defined as*

$$\hat{H}_k(T) = -\sum_{s\in\Sigma^k}\sum_{c\in\Sigma} p(c\mid s)\log p(c\mid s)$$

where $p(c \mid s)$ *means the probability of C if the previous k characters is S.*

Lemma 1. *For any string T and k⩾0, we have*

$$H_{k+1}(T) \le H_k(T)$$

Definition 3. *The* k-th order normalized entropy *of T is defined as*

$$H_k(T) = -\frac{1}{k}\sum_{s\in\Sigma^k} p(\mathrm{s})\log\, p(\mathrm{s})$$

978-0-7695-3896-9/09 $26.00 © 2009 IEEE
DOI 10.1109/ICCIT.2009.42

IEEE computer society

Authorized licensed use limited to: Universiti Malaysia Pahang. Downloaded on May 31,2022 at 05:55:52 UTC from IEEE Xplore. Restrictions apply.

The normalized entropy and the empirical entropy have the following relationship.

Theorem 1.

$$H_k(T) = \frac{1}{k}\left[\hat{H}_0(T) + \hat{H}_{k-1}(T)\right]$$

Proof.

$$\begin{aligned} H_k(T) &= -\sum_{c\in\Sigma}\sum_{s\in\Sigma^{k-1}} p(\mathrm{c,s})\log \mathrm{p(c,s)} \Big/ k \\ &= \left[H_1(T) + \sum_{s\in\Sigma^{k-1}} p(s) H_0(T_s) \right] \Big/ k \\ &= \left[\hat{H}_0(T) + \hat{H}_{k-1}(T) \right] \Big/ k \end{aligned}$$

Lemma 2. (Information inequality [2])
If $x \in \Sigma$, let p(x), q(x)be two probability mass functions. Then the relative entropy between them

$$D(p \,\|\, q) \geq 0$$

Lemma 3. (Chain rule for relative entropy [2])

$$\begin{aligned} D(p(x,y) \,|\, q(x,y)) = {} & D(p(x) \,|\, q(x)) \\ & + D(p(y \,|\, x) \,|\, q(y \,|\, x)) \end{aligned}$$

Lemma 4.

$$H(X) \leq \log|\Sigma|$$

where $|\Sigma|$ denotes the number of elements in the range of X, with equality if and only if X has a uniform distribution over $|\Sigma|$.

B. Self-index

For given text string $T_{1,n}$, the self-index can enhance its consumption of permanent storage and searching performance simultaneously by utilizing the regularities of its content. We denote $s(n)$ and $t(n)$ as the ideal space complexity and searching time that could be achieved in theory. Because $s(n)$ is undecidable for arbitrary data, any practical compression method is known to be only sub-optimal. For the self-index, we attempt to achieve the storage size limited by a number of bits

$$s(n) + \mathrm{o}(s(n)) \tag{1}$$

with a nearly-optimal asymptotic time bounds of

$$O(t(n)) \tag{2}$$

In fact, the formula that we use for quantitative analysis is an approximation to (1), namely

$$nH_k + M(T, \Sigma, k) \tag{3}$$

where $M(T, \Sigma, k)$ denotes the asymptotical number of bits used to store the empirical probabilities for the corresponding statistical model in T.

III. MODEL

A. Design

Definition 5. *In this paper, we simply define that data records is a set of n-tuples*

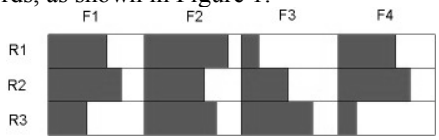
$$R = \{< f_1, \ldots, f_d > | \, f_1 \in D_1, \ldots, f_d \in D_d\}$$

where the domain D_i is a set of associated values. In addition, we presume that all domains can be conveyed by text medium.

We mainly perform three kinds of operations on R: append a record, extract a record with a given record id, retrieval particular records with given conditions [8]. For any file organization to store R, its performance of these operations should be considered primarily. Besides the performance, the storage occupancy is also another important aspect. Usually the designer needs to strike a proper balance between them according to the requirements of specific applications.

Now we focus on how to employ the self-index to store and query data records. Besides the searching functionalities, the self-index also supports extracting arbitrary portions from data records. This feature allows us to access the self-index just like the uncompressed text. In other words, the self-index provides the view of plain text to users with its indexed content. Thus, we first store R by a sort of the text file structure where the location of a record is context-free. This means that the location of the particular record is determined only by its unique id. We then build the self-index for the text. By doing so, the structured query will be converted to the problem of seeking and manipulating the occurrence of a group of patterns. On the other hand, the problem of these patterns matching will be fast solved by the self-index because of its approved searching performance.

Taking advantage of high compression of the self-index, we could design a kind of large text file organization by stuffing lots of marked characters in spite of storage consumption. In Section 4 the experiments will demonstrate that the increment of storage occupancy of the self-index after stuffing is comparable small to the uncompressed text. To build the mapping from the location of a text file to the corresponding record id, here we design a model to store data records, as shown in Figure 1.



Authorized licensed use limited to: Universiti Malaysia Pahang. Downloaded on May 31,2022 at 05:55:52 UTC from IEEE Xplore. Restrictions apply.

Figure 1. The organization of data records after stuffing delimiters. The light blocks denote the segments of delimiters, and the dark blocks denote the segments of original data.

The above model stores the records as a consecutive sequence. Each record contains the same number of fields. A record is of variable length only because some of its fields vary in length. To make the locations of records context-free, we insert a certain number of delimiters to the end of each field, except for using the original field separator. This will align the head positions of the same field of records. After stuffing these delimiters, the same field in all records will be represented with the equal width.

B. Upper Bound of Storage Occupancy

In this section, we will make a comparison of compressibility of a text between with and without stuffing. Let *T* and *T'*denote the text being stored as data records before and after stuffing in the above model. In Formula 3, the upper bound of storage occupancy is essentially determined by H_k.In the following, we will evaluate the impact of stuffing on H_k. For convenience, we do not distinguish the different types of delimiters, such as the field separator, the record separator, and the stuffing character. All delimiters will be denoted as C_s in the following discussion.

Definition 6. *Given a text T that stores R, we denote* $m = \max_{1\le i\le d} m_i$. *The number of substrings that equals* c_s^x *is a function f*(*x*):[1,*m*]→[1,|*T*|], *namely*

$$f(x) = c_i \qquad if\ x = i$$

,*where i*=1,2,...,*m*.

Lemma 5. *If we denote* α *as the number of segments containing* c_s *in T', and* β *as the number of* c_s *in T', namely* $\alpha = \sum_{i=1}^{m} f(i) = \sum_{i=1}^{m} c_i$, $\beta = \sum_{i=1}^{m} f(i)\cdot i = \sum_{i=1}^{m} c_i \cdot i$, *then*

$$\alpha = r \cdot d$$
$$\beta < r \cdot d \cdot m$$

Q.E.D.

As *T* and *T'* have the similar structure, the upper bound of the number of k-char substrings containing delimiter c_s can be calculated by the following lemma.

Lemma 6. *Let B*(*S*) *be the set of all k-char substring containing at least one consecutive* c_s *sequence in S, namely* $B(S) = \{s = Xc_s^n Y, s \in \Sigma^k, n > 0\}$. *For T, we have*

$$B(T) \le \alpha \cdot k$$

, and for T', we have

$$B(T') \le \alpha \cdot (k-1) + \beta$$

Proof. Either in *T* or in *T'*, there must be α segments of consecutive c_s . The number of substrings $s \in \Sigma^k$ that contain $c_s^n (n > 0)$ is

$$l+k-1$$

Thus, the number of substrings $s \in \Sigma^k$ that include c_s is not greater than (maybe duplicate)

$$B(T) \le \sum_{l=1}^{d} (l+k-1) f(l)$$

The following theorem gives an upper bound for the k-th order empirical entropy of *T'*.

Theorem 2.

$$H_k(T') \le \begin{cases} h(k) = \dfrac{n}{n+\beta-\alpha}\hat{H}_k(T) + [\alpha(k-1)+\beta] \\ \quad \cdot \log[(\sigma+1)^{k+1} - \sigma^{k+1}] - \dfrac{\beta}{n+\beta-\alpha}\log\dfrac{n}{\alpha}, & k \le d \\ h(d), & k > d \end{cases}$$

where n=|*T*|, *and* σ =|$\sum$|.

Proof.

The relative entropy between the $p'(c)$ and $p(c)$ is

$$\begin{aligned} & D(p'(c) \| p(c)) \\ &= \sum_c p'(c)\log p'(c) - \sum_c p'(c)\log p(c) \\ &= -\hat{H}_0(T') - \left[p'(c = c_s)\log p(c = c_s) + \sum_c p'(c \ne c_s)\log p(c \ne c_s) \right] \\ &= -\hat{H}_0(T') + \frac{\beta}{n+\beta-\alpha}\log\frac{n}{\alpha} + \frac{n}{n+\beta-\alpha}\hat{H}_0(T) \end{aligned}$$

(1) In the case of $1 \leqslant k \leqslant d$, and let s*=*s*·*c*, we have

$$\begin{aligned} & D(p'(c, s \in \Sigma^k) \| p(c, s \in \Sigma^k)) \\ &= D(p'(s^* \in \Sigma^{k+1}) \| p(s^* \in \Sigma^{k+1})) \\ &= -kH_{k+1}(T') - \sum_{s^*} p'(s^*)\log p(s^*) \\ &= -kH_{k+1}(T') - [\sum_{c_s \in s^*} p'(s^*)\log p(s^*) + \sum_{c_s \notin s^*} p'(s^*)\log p(s^*)] \\ &\le -kH_{k+1}(T') + [t(k-1)+\beta]\log[(\sigma+1)^{k+1} - \sigma^{k+1}] \\ &\quad + \frac{nk}{n+\alpha} H_{k+1}(T) \end{aligned}$$

From Lemma 2 and Lemma 3, we have

Authorized licensed use limited to: Universiti Malaysia Pahang. Downloaded on May 31,2022 at 05:55:52 UTC from IEEE Xplore. Restrictions apply.

$$0 \le D(p(c \mid s) \,\|\, p(c \mid s)) = D(p(c,s) \,\|\, p(c,s)) - D(p(c) \,\|\, p(c))$$

$$\le -kH_{k+1}(T') + [\alpha(k-1)+\beta]\log[(\sigma+1)^{k+1} - \sigma^{k+1}] + \frac{nk}{n+\beta-\alpha} H_{k+1}(T)$$

$$-[-\hat{H}_0(T') + \frac{\beta}{n+\beta-\alpha}\log\frac{n}{\alpha} + \frac{n}{n+\alpha}\hat{H}_0(T)]$$

$$= -\hat{H}_k(T') + \frac{n}{n+\beta-\alpha}\hat{H}_k(T) + [\alpha(k-1)+\beta]\log[(\sigma+1)^{k+1} - \sigma^{k+1}]$$

$$-\frac{\beta}{n+\beta-\alpha}\log\frac{n}{\alpha}$$

(2) In the case of k>d, from Lemma 1,we have

$$H_k(T') \ge H_d(T')$$

In Formula 3, if we substitute *T'* for *T* and *H*(*T'*) for *H*(*T*), we can get the upper bound of storage for any kind of self-indexes.

IV. EXPERIMENTAL RESULTS

Our experiment compares two aspects: storage occupancy and performance. We select the FM-index [3] as the representation of self-indexes. The datasets in the experiments are collected from different sources to cover various types of applications, as shown in Table 1.

We compare our method against Microsoft SQL Server 2000, as well as against GNU grep 2.5.1 (via Cygwin). The former is a popular database management system, while the latter is a tool for the scan-based text retrieval. Our experiments ran on an Intel Genuine T2300 (dual-core) processor at 1.66GHz, with 2MB L2 cache, 512MB DDR2 RAM, and 60GB IDE disk. The indexing program, including the FM-index and our extension, ran on the Window XP SP2 via the Cygwin library. The default settings of the FM-index in the experiments are used; that is, superbucket size = 16, bucket size = 512, and frequency of marked characters = 0.02. In addition, we set the block size = 10MB and overlapping length = 1KB.

A. Storage

We compare the storage occupancy by different ways of storing data records, as shown in Table 2. The first two rows of the table give the size of texts before and after stuffing, respectively. The increments in size are obvious. The next two rows show the size of the FM-index built for the above texts. Compared to the pure texts, the size of the FM-index is slightly increased. The last row shows the space used in the SQL Server for the same texts. It consumes much more space than does the FM-index even without additional indexes.

B. Performance

The experiment on performance includes three operations: construct, recover, and retrieval. The results are reported in Table 3.

In general, we found that the FM-index is able to achieve comparable performances on constructing and recovering with the SQL Server. The query performance of the FM-index is much better than the grep, but worse than the SQL Server, especially for lager datasets. This means the query performance needs to be further optimized. Anyhow, our solution provides a new tradeoff choice between permanent space and running time for users.

TABLE I. DATA FILES USED IN THE EXPERIMENTS.

Dataset	Records	Fields	Length of Records Max/Ave/Min	Content
ACCOUNT[1]	1,403,736	8	78/68/52	Account chart
SOCIETY[2]	11,018	123	376/196/36	Homicide records in Chicago
NURSING[3]	59,156	16	472/279/172	Nursing home inspection
WILDBIRD[4]	15,705	57	125/81/73	Wild birds cases of West Nile virus

TABLE II. THE COMPARISON OF STORAGE OCCUPANCY USING DIFFERENT METHODS OF FILE ORGANIZATIONS. THE MDF IS THE DATABASE FORMAT USED BY THE SQL SERVER.

Size(bytes)		ACCOUNT	SOCIETY	NURSING	WILDBIRD
Text	***Original***	106,703,160	4,418,279	27,646,363	2,172,006
	Stuffing	122,125,032	17,066,914	32,062,552	3,879,135
Compressed	***FM-index for Original***	11,317,830	965,940	2,380,449	191,892
	FM-index for Stuffing	11,532,983	1,822,508	2,706,330	264,681
MDF(SQL Server)		149,291,008	12,320,768	69,533,696	5,636,096

[1] National Audit Office of the People's Republic of China, http://www.cnao.gov.cn/main/index.htm
[2] Homicide In Chicago 1870-1930, http://homicide.northwestern.edu/download/
[3] U.S. Department of Health \& Human Services, http://www.medicare.gov/Download/DownloadDB.asp
[4] National Atlas of the United States, http://www.nationalatlas.gov/mld/w01avit.html

Authorized licensed use limited to: Universiti Malaysia Pahang. Downloaded on May 31,2022 at 05:55:52 UTC from IEEE Xplore. Restrictions apply.

TABLE III. THE COMPARISON OF PERFORMANCE. THE RUNNING TIME OF THE FM-INDEX TOOL AND GREP WAS MEASURED BY MILLISECONDS, WHILE THE SQL SERVER IS MEASURED BY SECONDS DUE TO TECHNICAL REASONS. THE QUERIES USEDIN RETRIEVAL OPERATIONS ARE GENERATED RANDOMLY FROM THE DATA FILE. WE EXECUTE 100 SAME QUERIES FOR EVERY TOOL AT ONE TIME.

Time(s)		ACCOUNT	SOCIETY	NURSING	WILDBIRD
Construct	***SQL***	28	3	2	1
	FM	65.015	8.468	21.843	2.125
Recover	***SQL***	24	2	2	<1
	FM	28.281	4.312	6.312	0.890
Retrieval	***SQL***	8.2	<0.1	<0.1	<0.1
	FM	154.323	0.140	0.216	0.097
	Grep	570.734	35.016	217.140	16.265

V. CONCLUSION

In this paper we have presented an approach to building the self-index for data records. After introducing the design and implementation of our approach, we make a theoretical analysis of its performance and storage requirements. At last, we compare the performance with similar tools by experiments. In the future we will optimize the performance for lager datasets. Because the data in the same field has the similar distribution, we will try to build the data by individual fields. This will possibly enhance the performance.

ACKNOWLEDGMENT

We would like to thank all reviewers for their helpful comment on this paper. The work is supported by Beijing Municipal Commission of Education(KM200911417002).

REFERENCES

[1] *G. Blelloch, "Introduction to data compression," Book Introduction to data compression, Series Introduction to data compression, ed., 2000.*

[2] *T. Cover and J. Thomas, Elements of Information Theory, John Wiley & Sons, Inc., 2001.*

[3] *P. Ferragina and G. Manzini, "Opportunistic data structures with applications," Proc. the 41st IEEE Symposium on Foundations of Computer Science, 2000, pp. 390-398.*

[4] *G.A. Grossi R., Vitter J.S., "An Algorithmic Framework for Compression and Text Indexing," http://www.cs.purdue.edu/~jsv/Papers/GGV05.algorithmicframework .pdf, 2007.*

[5] *D. Grossman, et al., "Integrating Structured Data and Text: A relational approach," Journal of the American Society for Information Science, vol. 48, no. 2, 1997, pp. 122-132.*

[6] *G. Manzini, "An analysis of the Burrows-Wheeler transform," Journal of the ACM (JACM), vol. 48, no. 3, 2001, pp. 407-430.*

[7] *G. Navarro and V. Mäkinen, "Compressed Full-Text Indexes," ACM Computing Surveys, vol. 39, no. 1, 2007.*

[8] *R. Ramakrishnan and J. Gehrke, Database management systems (3rd ed.), McGraw-Hill Science/Engineering/Math, 2002.*

Authorized licensed use limited to: Universiti Malaysia Pahang. Downloaded on May 31,2022 at 05:55:52 UTC from IEEE Xplore. Restrictions apply.